# Computer Science Foundation Exam

## December 14, 2012

## Section I B

## COMPUTER SCIENCE

**NO books, notes, or calculators may be used,**
**and you must work entirely on your own.**

## SOLUTION

| Question # | Max Pts | Category | Passing | Score |
|---|---|---|---|---|
| 1 | 10 | ANL | 7 | |
| 2 | 10 | DSN | 7 | |
| 3 | 10 | DSN | 7 | |
| 4 | 10 | ALG | 7 | |
| 5 | 10 | ALG | 7 | |
| TOTAL | 50 | | | |

**You must do all 5 problems in this section of the exam.**

**Problems will be graded based on the completeness of the solution steps and <u>not</u> graded based on the answer alone. Credit cannot be given unless all work is shown and is readable. Be complete, yet concise, and above all <u>be neat</u>.**

**1)** (10 pts) ALS (Algorithm Analysis)

(a) (6 pts) List the best case, worst case and average case run-times of each of the following algorithms/operations in terms of their input size, n:

(i) Inserting an item into a Linked List of n elements

best case: **O(1)** average case: **O(n)** worst case: **O(n)**

(ii) A Quick Sort of n elements

best case: **O(nlgn)** average case: **O(nlgn)** worst case: **O($n^2$)**

(iii) Searching for an element in a binary tree

best case: **O(1)** average case**: O(lg n)** worst case: **O(n)**

**Grading: 2 pts per question. Give two points if all three are correct, one point if at least one is correct, and 0 pts otherwise, for all three parts.**

(b) (4 pts) Consider the recurrence relation T(n) = T(n-1) + $n2^n$, for n > 1 and T(1) = 2. Rewrite the value of T(n) utilizing a summation so T(n) is expressed without any reference to T(x), for any value x. **Please leave your final answer as a summation and DO NOT ATTEMPT to solve the sum.**

Iterating one step, we find: $T(n) = T(n-2) + (n-1)2^{n-1} + n2^n$.
Iterating again, we find: $T(n) = T(n-3) + (n-2)2^{n-2} + (n-1)2^{n-1} + n2^n$.

In general, iterating any recurrence of this type leads to a summation. In particular, if we let f(n) = $n2^n$, then the iteration process ends with:

$$T(n) = T(1) + \sum_{i=2}^{n} i2^i$$

If we note that T(1) = 1 x $2^1$, we can add T(1) into the summation to yield: $T(n) = \sum_{i=1}^{n} i2^i$.

**Answer: $\boldsymbol{T(n) = \sum_{i=1}^{n} i2^i}$**

**Grading: 1 point for summation, 1 point for bounds on sum, 2 pts for the function being summed, in terms of the summation index**

**2)** (10 pts) DSN (Recursive Algorithms – Binary Trees)

Write a **recursive** function that will return the number of nodes in a binary tree that contain a particular value. Your function will take in a pointer to the root of the binary tree as well as the value for which to search. The prototype for the function and the binary tree struct are given to you below. Complete the function.

```
struct treeNode {
    int data;
    struct treeNode *left;
    struct treeNode *right;
};

int numOccurrences(struct treeNode* root, int value) {

    if (root == NULL)              // 1 pt
        return 0;                  // 1 pt

    int sum = 0;                   // 1 pt
    if (root->data == value)       // 1 pt
        sum++;                     // 1 pt

    sum = sum + numOccurrences(root->left, value)      // 2 pts
              + numOccurrences(root->right, value);    // 2 pts

    return sum;                                        // 1 pt
}
```

**3)** (10 pts) DSN (Linked Lists)

Imagine using a linked list to store a large integer. In particular, each node of the linked list stores a single digit with the least significant digit being stored first. For example, the number 1387 would be stored in a linked list of length four storing the digits 7, 8, 3, and 1, respectively. Complete the function below that compares to large integers stored in this fashion. In particular, if the first number is strictly less than the second number, return -1. If they are equal, return 0, Otherwise, return 1. Although it's not a requirement, it's probably easiest to write this function recursively. Use the struct and function prototype provided below.

```
struct node {
    int data;
    struct node *next;
};

int intcmp(struct node *ptrA, struct node* ptrB) {

    if (ptrA == NULL && ptrB == NULL) // 1 pt
        return 0;

    if (ptrA == NULL)                    // 1 pt
        return -1;

    if (ptrB == NULL)                    // 1 pt
        return 1;

    int ans = intcmp(ptrA->next, ptrB->next); // 3 pts

    if (ans != 0)                        // 1 pt
        return ans;

    if (ptrA->data < ptrB->data)         // 1 pt
        return -1;
    else if (ptrA->data > ptrB->data)    // 1 pt
        return 1;
    else
        return 0;                        // 1 pt




}
```

**4)** (10 pts) ALG (Tracing) Consider the following function:

```
int f(int array[], int length, int target) {

    int i=0, j=0, sum=0, cnt=0;

    while (j < length) {
        if (sum < target) {
            sum += array[j];
            j++;
        }
        else if (sum > target) {
            sum -= array[i];
            i++;
        }
        else {
            cnt++;
            sum -= array[i];
            i++;
        }
    }

    if (sum == target) cnt++;

    return cnt;
}
```

(a) (3 pts) If array stores the elements 2, 3, 3, 2, 5, 4, 1, 3, 6, 8, 2, 3, 4, 4, 2, 2, what would the return value of the function call f(array, 16, 8) be?

**The else clause triggers 6 times, for the following ordered pairs (i,j): (0,2), (1,3), (5,7), (9,9), (12,13), (13, 15).**

**6 (3 pts correct answer, 2 pts for off by 2 or less, 1 pt for off by 4 or less, 0 otherwise)**

(b) (4 pts) Give a concise description of what f1 calculates.

**It calculates the number of contiguous subsequences (sets of consecutive numbers) in the array that add up to the target exactly. (2 pts consecutive numbers, 1 pt sum, 1 pt equals target)**

(c) (3 pts) Let n be the length of the input array to f. What is the run time of f in terms of n?

**Both i and j can be iterated at most n times, and in each loop iteration, one of the two of them are incremented. It follows that the loop body, which has all constant time statements, runs at most 2n times. Thus, the desired run time is O(n).**

**<u>O(n)</u> (3 pts all or nothing)**

**5)** (10 pts) ALG (Sorting)

(a) (4 pts) Consider running a Merge Sort on the array below. What would the contents of the array be right BEFORE the last Merge operation?

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Values | 13 | 2 | 8 | 7 | 14 | 6 | 19 | 1 |

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Values | **2** | **7** | **8** | **13** | **1** | **6** | **14** | **19** |

**Grading: 1 pt for 2, 7, 8 and 13 staying in indexes 0-3, 1 pt for 1, 6, 14 and 19 staying in indexes 4-7. 2 pts for the ordering**

(b) (3 pts) Consider running an insertion sort on the array below. How many swaps would be performed total while the algorithm ran?

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Values | 13 | 2 | 8 | 7 | 14 | 6 | 19 | 1 |

WE perform 1 swap for each inversion in the array. Here are all the inversions in the array: (13, 2), (13, 8), (13, 7), (8, 7), (13, 6), (8, 6), (7, 6), (14, 6), (13, 1), (2, 1), (8, 1), (7, 1), (14, 1), (6, 1), and (19, 1).

**15 (3 pts correct answer, 2 pts off by 2 or less, 1 pt off by 6 or less, 0 pts otherwise)**

(c) (3 pts) Given the array below, which element, 8 or 83 would be a better pivot element for running the Partition in Quick Sort? Why? (Note: By better pivot element, we mean a choice of pivot that's likely to reduce the run time of the algorithm.)

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Values | 13 | 2 | 8 | 7 | 14 | 6 | 19 | 1 | 99 | 5 | 57 | 44 | 3 | 31 | 83 |

**8 would be better because it's closer to the median value in the array. In general, Quick Sort runs better if the pivot element chosen for the Partition splits the array into two roughly equal sized sets. The value 8 is one value away from the median of the array, 13. Thus, it's a better choice of pivot than 83, the second largest value in the array. (1 pt answer, 2 pts why)**